



ALL PROGRAMMING MODELS ARE WRONG BUT SOME ARE USEFUL: IDENTIFYING PRODUCTIVE ABSTRACTIONS FOR EXASCALE SIMULATION

Jeff Hammond

Exascale CoDesign, Data Center Group, Intel Corporation

Acknowledgements:

Michael Klemm, Tim Mattson, Rob van der Wijngaart, Alex Duran, Jim Cownie,
Alexey Kukanov, Mike Kinsner, Ben Ashbaugh, NWChem team at PNNL,
Tom Scoglund and the rest of the RAJA team at LLNL, CodePlay SYCL team, ...

Notices and Disclaimers

© 2018 Intel Corporation. Intel, the Intel logo, Xeon and Xeon logos are trademarks of Intel Corporation in the U.S. and/or other countries

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. . Performance varies depending on system configuration. **No computer system can be absolutely secure.** Check with your system manufacturer or retailer or learn more at intel.com/performance/datacenter.

Some results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

The cost reduction scenarios described are intended to enable you to get a better understanding of how the purchase of a given Intel based product, combined with a number of situation-specific variables, might affect future costs and savings. Circumstances will vary and there may be unaccounted-for costs related to the use and deployment of a given product. Nothing in this document should be interpreted as either a promise of or contract for a given level of costs or cost reduction.

Intel processors of the same SKU may vary in frequency or power as a result of natural variability in the production process.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804.

Performance estimates were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to www.intel.com/benchmarks

*Other names and brands may be claimed as the property of others.

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



Additional Disclaimer

I am not an official spokesman for any Intel products. I do not speak for my collaborators, whether they be inside or outside Intel.

I work on system pathfinding and workload analysis, not software products. I am not a developer of Intel software tools.

You may or may not be able to reproduce any performance numbers I report, but the code is on GitHub and I will provide anything else you need to attempt to reproduce my results.

Hanlon's Razor (blame stupidity, not malice).

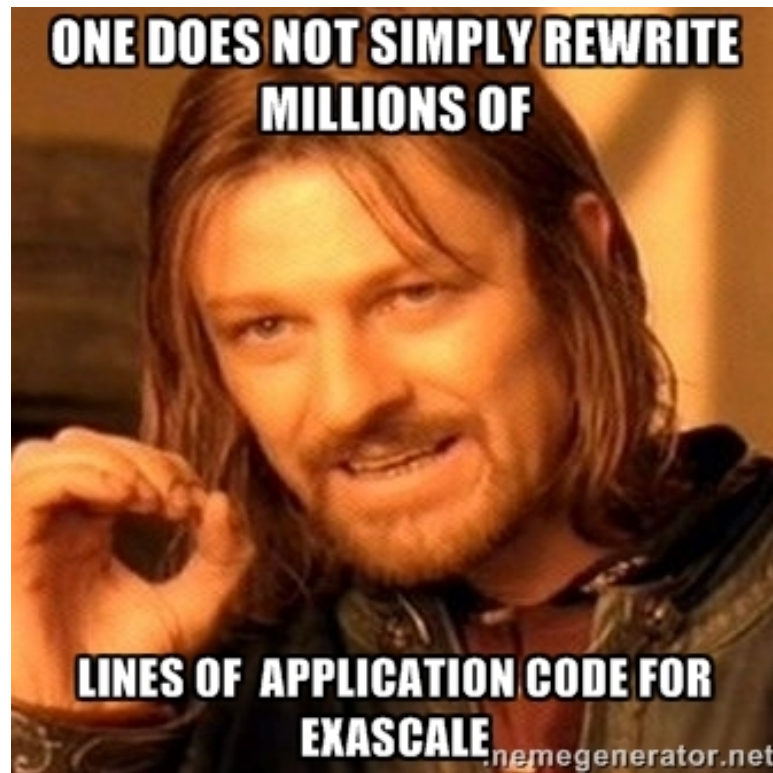
Outline

- Background and context
- Parallel Research Kernels
- C++ parallelism
- NWChem and experiences with OpenMP

HPC software design challenges (2014)

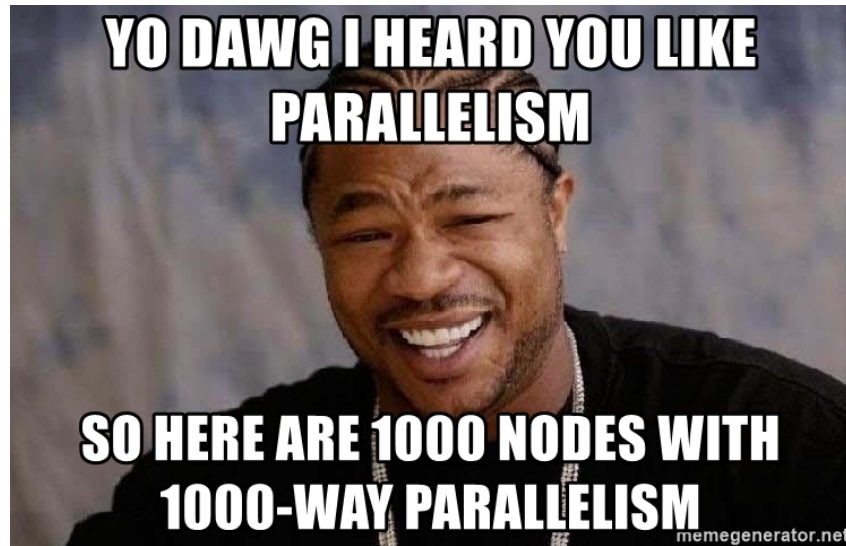
- To MPI or not to MPI...
- One-sided vs. two-sided?
- Does your MPI/PGAS need a +X?
- Static vs. dynamic execution model?
- What synchronization motifs maximize performance across scales?

Application programmers can afford to rewrite/redesign applications zero to one times every 20 years...



HPC software design challenges (2018)

- Intranode parallelism is growing much faster than internode...
- Intranode parallelism is far more diverse than internode parallelism.
 - After ~20 years, internode behavior is converged to some subset of MPI-3.
 - Big Cores, Little Cores, GPU, FPGA all require (very) different programming models.



How do we *measure* productivity+performance+portability?

PARALLEL RESEARCH KERNELS

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



Programming model evaluation

Standard methods

- NAS Parallel Benchmarks
- Mini Applications
(e.g. Mantevo, LULESH)
- HPC Challenge

There are numerous examples of these on record, covering a wide range of programming models, but is source available and curated?

What is measured?

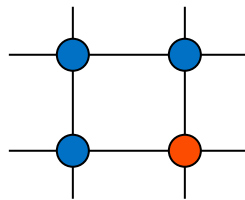
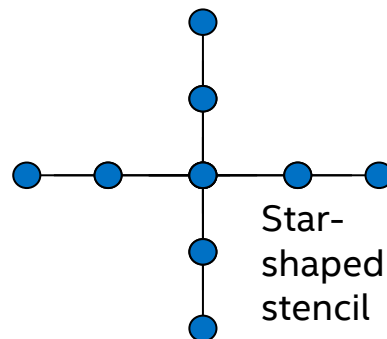
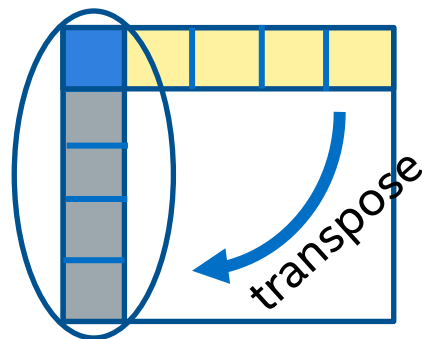
- Productivity (?), elegance (?)
- Implementation quality
(runtime or application)
- Asynchrony/overlap
- Semantics:
 - Automatic load-balancing (AMR)
 - Atomics (GUPS)
 - Two-sided vs. one-sided, collectives

Goals of the Parallel Research Kernels

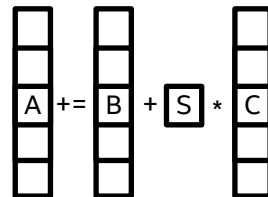
1. **Universality:** Cover broad range of performance critical application patterns.
2. **Simplicity:** Concise pencil-and-paper definition and transparent reference implementation. *No domain knowledge required.*
3. **Portability:** Should be implementable in any sufficiently general programming model.
4. **Extensibility:** Parameterized to run at any scale. Other knobs to adjust problem or algorithm included.
5. **Verifiability:** Automated correctness checking and built-in performance metric evaluation.
6. ~~Hardware benchmark:~~ No! Use HPCChallenge, Xyz500, etc. for this.

Outline of PRK Suite

- **Dense matrix transpose**
- Synchronization: global
- **Synchronization: point to point**
- **Scaled vector addition**
- Atomic reference counting
- Vector reduction
- Sparse matrix-vector multiplication
- Random access update
- **Stencil computation**
- Dense matrix-matrix multiplication
- Branch
- Particle-in-cell



$$A_{i,j} = A_{i-1,j} + A_{i,j-1} - A_{i-1,j-1}$$



Static kernels

Language	Seq.	OpenMP	MPI	PGAS	Threads	Others?
C89	✓	✓	Many	SHMEM		
C99/C11	✓	✓✓✓		UPC	✓	Cilk, ISPC
C++17	✓	✓✓✓		Grappa	✓	Kokkos, RAJA, TBB, PSTL, SYCL, Boost.Compute, OpenCL, CUDA...
Fortran	✓	✓✓✓		coarrays		“pretty”, OpenACC
Python	✓					Numpy
Chapel	✓			✓		

✓✓✓ = Traditional, task-based, and target are implemented identically in Fortran, C and C++.

Additional language support includes Rust, Julia, and Matlab/Octave.

<> Code

! Issues 30

🔗 Pull requests 4

📁 Projects 0

📖 Wiki

📊 Insights

⚙️ Settings

This is a set of simple programs that can be used to explore the features of a parallel platform.

Edit

<https://groups.google.com/forum/#!for...>

parallel-programming

parallel-computing

c

c-plus-plus

mpi

fortran2008

python3

julia

pgas

openmp

shmem

coarray-fortran

travis-ci

charmplusplus

threading

tbb

kokkos

opencl

sycl

boost

Manage topics

📄 2,801 commits

🔗 11 branches

📦 6 releases

👤 18 contributors

Branch: master ▾


New pull request

Create new file

Upload files

Find file

Clone or download ▾

 jeffhammond remove STL usage from OpenMP codes (#363)

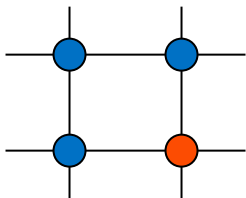
Latest commit 46741c0 2 days ago

📁 .github	try out issue/PR templates	7 months ago
📁 AMPI	MPI comm are not C int	8 months ago
📁 C1z	deprecate Cilk (#357)	2 months ago
📁 CHARM++	Fix spacing in Charm++ Stencil Makefile (#362)	a month ago
📁 Cxx11	remove STL usage from OpenMP codes (#363)	2 days ago

Synch point-to-point

```
for i in range(1,m):  
    for j in range(1,n):  
        A[i][j] = A[i-1][j]  
                + A[i][j-1]  
                - A[i-1][j-1]
```

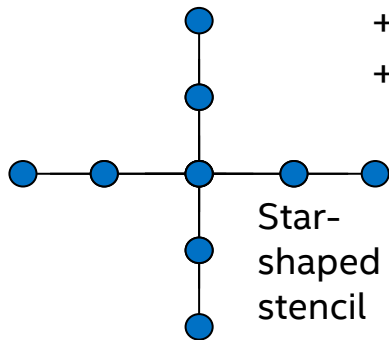
$$A[0][0] = -A[m-1][n-1]$$



$$A_{i,j} = A_{i-1,j} + A_{i,j-1} - A_{i-1,j-1}$$

- Proxy for discrete ordinates neutron transport; much simpler than SNAP or Kripke.
- Proxy for dynamic programming, which is used in sequence alignment (e.g. PairHMM).
- Wraparound to create dependency between iterations.

Stencil

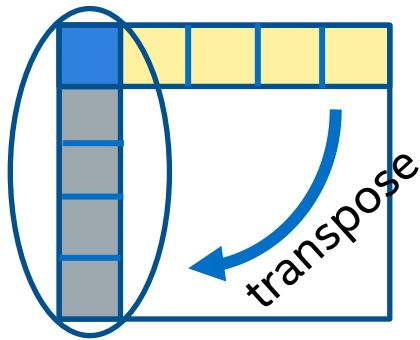
$$\begin{aligned} B[2:n-2, 2:n-2] &+= W[2,2] * A[2:n-2, 2:n-2] \\ &+ W[2,0] * A[2:n-2, 0:n-4] \\ &+ W[2,1] * A[2:n-2, 1:n-3] \\ &+ W[2,3] * A[2:n-2, 3:n-1] \\ &+ W[2,4] * A[2:n-2, 4:n-0] \\ &+ W[0,2] * A[0:n-4, 2:n-2] \\ &+ W[1,2] * A[1:n-3, 2:n-2] \\ &+ W[3,2] * A[3:n-1, 2:n-2] \\ &+ W[4,2] * A[4:n-0, 2:n-2] \end{aligned}$$


- Proxy for structured mesh codes. 2D stencil to emphasize non-compute.
- Supports arbitrary radius star and square stencils via code generator for C11 and C++ models, which was inspired by OpenCL.

Transpose

```
for i in range(order):  
    for j in range(order):  
        B[i][j] += A[j][i]  
        A[j][i] += 1.0
```

- Proxy for 3D FFT, bucket sort...
- Local transpose of square tiles supports blocking to reduce TLB pressure.



C++ AND PARALLELISM

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



I study molecular dynamics, but to tell the truth I am interested more in the dynamics than in the molecules, and I care most about questions of principle.

Phil Pechukas, Columbia University Chemical Physics Professor

I study C++ parallelism, but to tell the truth I am interested more in the parallelism than in the C++, and I care most about questions of practice.

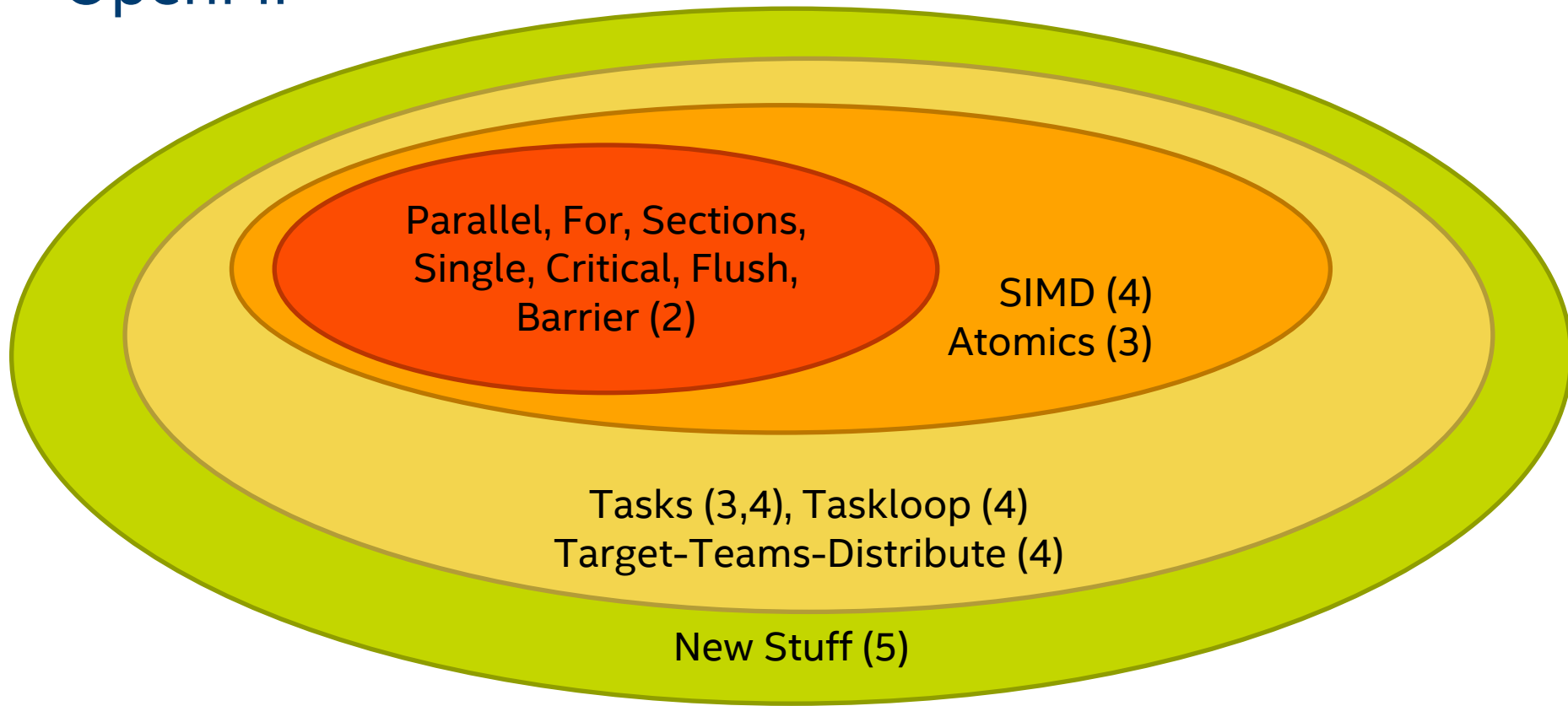
Why C++ parallelism?

- C++ is a kitchen sink language – it has pretty much every feature that exists in programming languages (other than simplicity and orthogonality).
- Used across essentially all markets/domains where parallelism or performance matter.
 - Fortran and Rust usage domain-specific.
 - Interpreted languages do not satisfy performance requirements.
- C++ can be extended to do all sorts of things within the language itself. Variadic templates for fun and profit!
- Mattson's Law: No new languages!

Overview of Parallel C++ models

- TBB (Intel OSS) - parallel threading abstraction for CPU*.
- KOKKOS (Sandia) – parallel execution and data abstraction for CPU and GPU architectures (OpenMP, Pthreads, CUDA, ...).
- RAJA (Livermore) – parallel execution for CPU and GPU architectures (OpenMP, TBB, CUDA, ...). CHAI adds GPU data abstraction.
- PSTL (ISO standard) – parallel execution abstraction for CPU architectures; designed for future extensions for GPU, etc. (e.g. Thrust and HPX).
- SYCL (Khronos standard) - parallel execution and data abstraction that extends the OpenCL model (supports CPU, GPU, FPGA, ...).

OpenMP



Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.

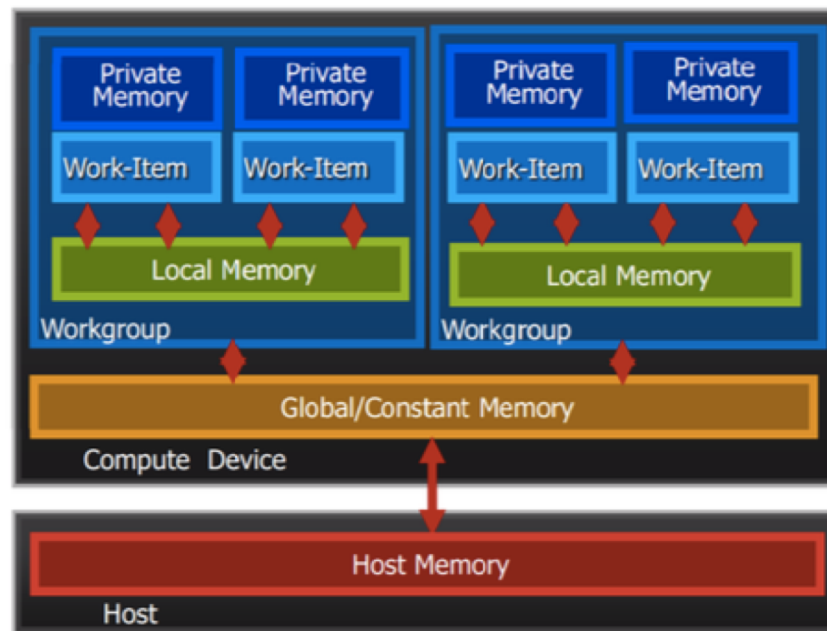
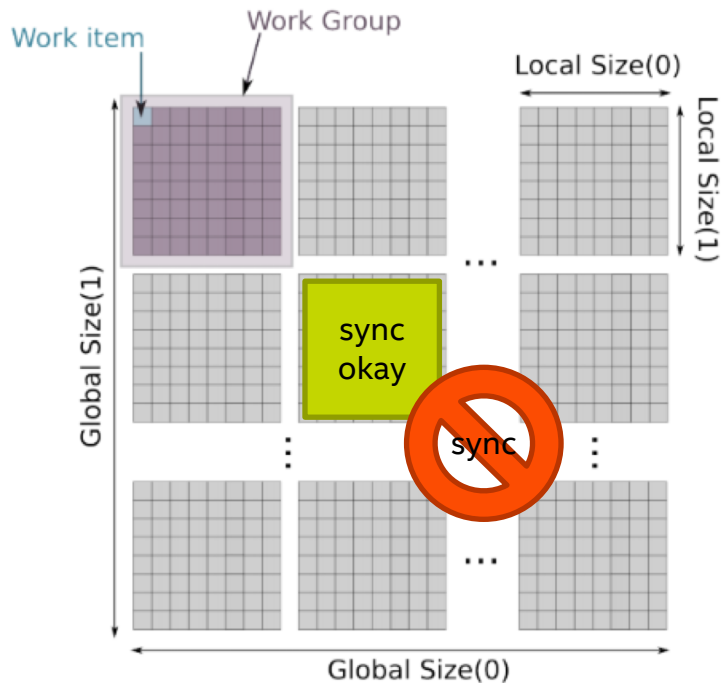


An Introduction to the OpenCL Programming Model

Jonathan Tompson*
NYU: Media Research Lab

Kristofer Schlachter†
NYU: Media Research Lab

OpenCL 2 is a bit more complicated, but doesn't change the execution model.



Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



SYCL

- Khronos standard based on C++11 and OpenCL.
- Retains the OpenCL execution model: `work_groups` + `work_items`.
 - May require extensions for SIMD exec to support forward deps.
- Single-source programming model (may be >1 compiler passes).
- Eliminates the painful boilerplate code associated with OpenCL.
- OpenCL interoperability (e.g. OpenCL linear algebra libraries).

All experiments use the CodePlay* ComputeCpp implementation based on Clang/LLVM that generates SPIR-V.

Model	for	for ^N	reduce	scan	Hierarchy/Composition
TBB::parallel	Y	Y	Y	Y	Threads
C++17 PSTL	Y	N [^]	Y	Y	Threads+SIMD; new?
RAJA	Y	Y	Y	Y	Threads+SIMD; CUDA
KOKKOS	Y	Y	Y	Y	Team+Thread+SIMD
Boost.Compute	Y	N ^{*^}	Y	Y	N
SYCL	Y	3	N	N	Group(+Subgroup)+Item
OpenCL 1.x	Y	3	N	N	Group+Item
OpenMP 5	Y	Y	Y	Y	Y ^{**}

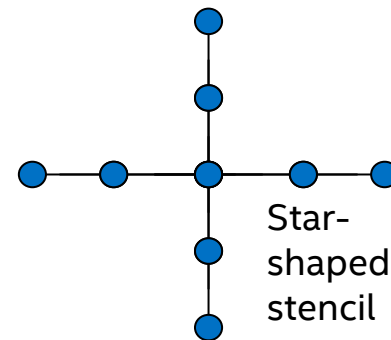
- * Boost.Compute supports embedded OpenCL, which in turn exposes 3D loop nests.
- ** OpenMP nested parallelism is unpleasant. You can nest “parallel for” or switch paradigms to “taskloop” and give up on accelerator support.
- ^ One can always implement a collapsed N-d loop but that adds div/mod to loop body.

HPC-like vs STL-like vs OpenCL-like

- TBB HPC-like
 - Nested, blocked forall w/ affinity control and load-balancing
- RAJA
 - Nested, blocked, permuted forall w/ fine-grain policy control.
- KOKKOS
 - Nested, blocked, permuted forall.

- C++17 (parallel STL) STL-like
 - Parallel STL evolving towards GPU etc.
- Boost.Compute
 - Effectively parallel STL over OpenCL.
- SYCL OpenCL-like
 - OpenCL execution model
 - Parallel STL over SYCL exists.

The HPC-like models capture the popular OpenMP idioms while hiding complexity.



PERFORMANCE EXPERIMENTS

<https://github.com/ParRes/Kernels/tree/master/Cxx11>

Please contact the author if you are interested in performance data produced by the PRKs.

WAVEFRONT PARALLELISM

Optimization Notice

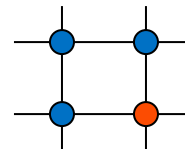
Copyright © 2015, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



Wavefront Parallelism

```
// sequential C implementation
for (int i=1; i<m; ++i) {
    for (int j=1; j<n; ++j) {
        A[i][j] = A[i-1][j] + A[i][j-1] - A[i-1][j-1];
    }
}
```

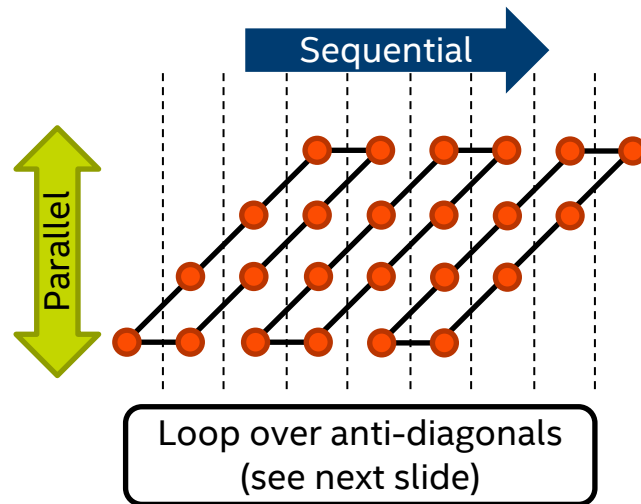
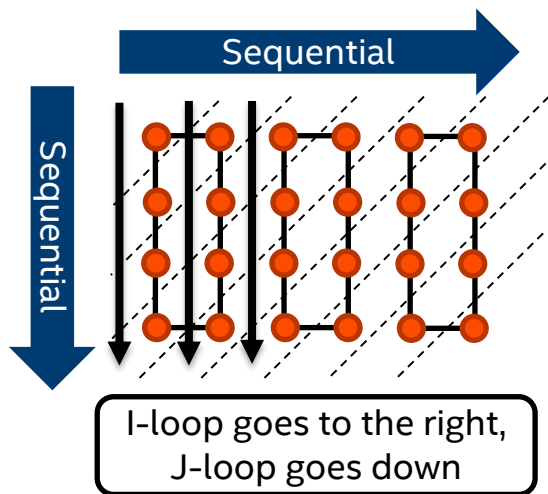


$$A_{i,j} = A_{i-1,j} + A_{i,j-1} - A_{i-1,j-1}$$

This *pattern* appears in a range of applications:

- Deterministic neutron transport (DOE-NNSA mission science)
- Smith-Waterman/PairHMM (bioinformatics)
- Dynamic programming
- Linear algebra (e.g. NAS LU benchmark)

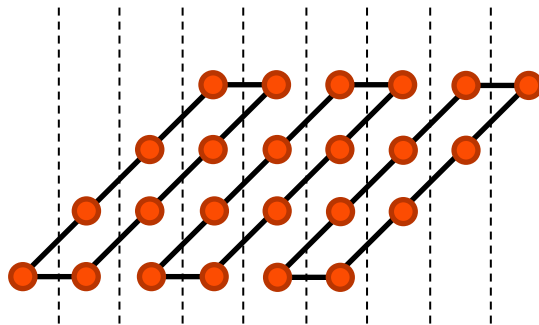
Changing the iteration space exposes parallelism



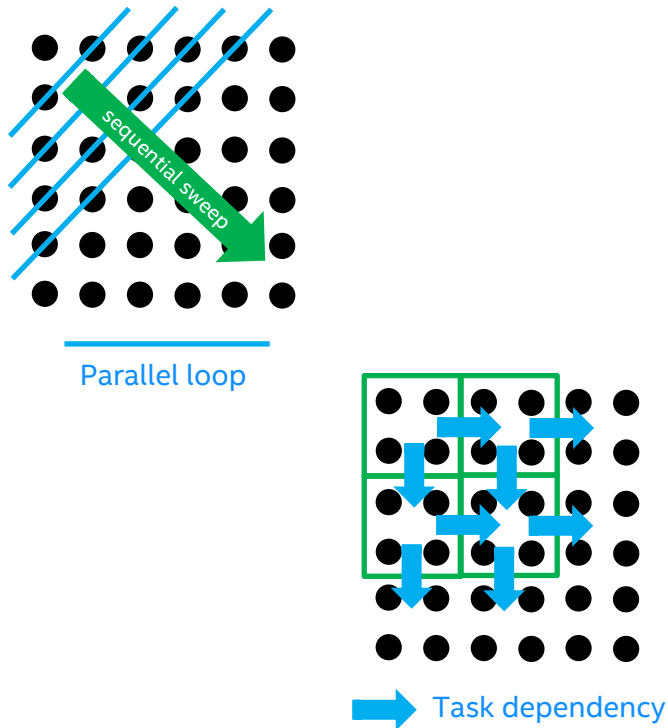
OpenMP inner-loop parallelism

```
// sequential loop
for (int i=2; i<=2*n-2; ++i) {
    int start = max(2,i-n+2);
    int stop  = min(i,n);
    #pragma omp for simd
    for (int j=start; j<=stop; ++j) {
        const int x = i-j+1;
        const int y = j-1;
        A[x][y] = A[x-1][y]
                  + A[x][y-1]
                  - A[x-1][y-1];
    }
    // implicit barrier (required)
}
```

- Very low parallel efficiency once data spills private cache.
- CPU SIMD doesn't work because data access is non-contiguous.



Amortizing synchronization overheads



- Sequential execution requires no synchronization.
- Formally, there are $O(n^2)$ element-wise dependencies.
- Antidiagonal implementation uses $O(n)$ barriers to enforce deps.
- Hyperplane amortizes barriers across many antidiagonals: $O(n/\text{unroll})$ barriers.
- Task-based has $O(n^2/\text{block}^2)$ dependencies.

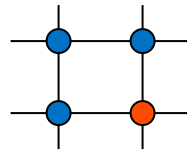
OpenMP hyperplane parallelism

```
#pragma omp parallel
for (int i=2; i<=2*(nb+1)-2; i++) {
    #pragma omp for
    for (int j=std::max(2,i-(nb+1)+2); j<=std::min(i,nb+1); j++) {
        const int ib = nc*(i-j)+1;
        const int jb = nc*(j-2)+1;
        for (int ii=ib; ii<std::min(m,ib+nc); ii++) {
            for (int jj=jb; jj<std::min(n,jb+nc); jj++) {
                A[ii][jj] = A[ii-1][jj] + A[ii][jj-1] - A[ii-1][jj-1];
            }
        }
    }
}
```

This is only implemented for square grids to keep the polyhedral arithmetic simpler.

OpenMP task-based parallelism

```
#pragma omp parallel
#pragma omp master
for (int i=1; i<m; i+=mc) {
    for (int j=1; j<n; j+=nc) {
        #pragma omp task depend(in:grid[i-mc][j],grid[i][j-nc]) \
                                depend(out:grid[i][j])
        for (int ii=i; ii<std::min(m,i+mc); ii++) {
            for (int jj=j; jj<std::min(n,j+nc); jj++) {
                A[ii][jj] = A[ii-1][jj] + A[ii][jj-1] - A[ii-1][jj-1];
            }
        }
    }
}
#pragma omp taskwait
```

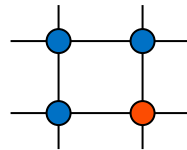


$$A_{i,j} = A_{i-1,j} + A_{i,j-1} - A_{i-1,j-1}$$

OpenMP “doacross” parallelism

The Intel OpenMP already has an improved implementation of this feature...

```
#pragma omp for collapse(2) ordered(2)
for (int i=0; i<ib; i++) {
    for (int j=0; j<jb; j++) {
        #pragma omp ordered depend(sink: i-1,j) depend(sink: i,j-1)
        for (int ii=i; ii<std::min(m,i+mc); ii++) {
            for (int jj=j; jj<std::min(n,j+nc); jj++) {
                A[ii][jj] = A[ii-1][jj] + A[ii][jj-1] - A[ii-1][jj-1];
            }
        }
    }
    #pragma omp depend(source)
}
```



$$A_{i,j} = A_{i-1,j} + A_{i,j-1} - A_{i-1,j-1}$$

Summary

- Parallel C++ effectively hides the complexity of underlying models like OpenMP and OpenCL without introducing any overhead (on CPUs).
- Implementation differences between OpenMP and TBB schedulers show places where OpenMP runtimes can be improved.
- PSTL (based on TBB in Intel's implementation) works well on CPUs but is limited by STL semantics. PSTL portability requires evolution of C++ towards HPX, Thrust...
- SYCL provides a modern C++ abstraction and single-source compilation on top the OpenCL execution model.
- Task-based parallelism has a good ROI for wavefront algorithms.

NWCHEM

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved.

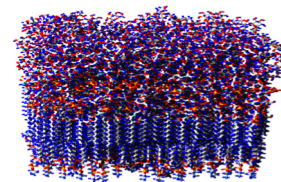
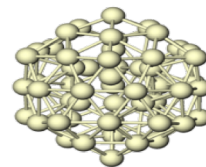
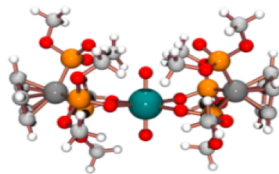
*Other names and brands may be claimed as the property of others.



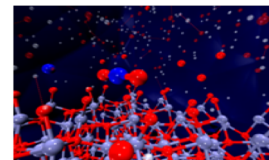
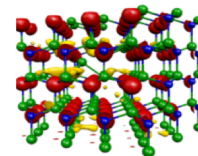
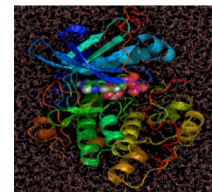


NWCHEM

HIGH-PERFORMANCE COMPUTATIONAL
CHEMISTRY SOFTWARE

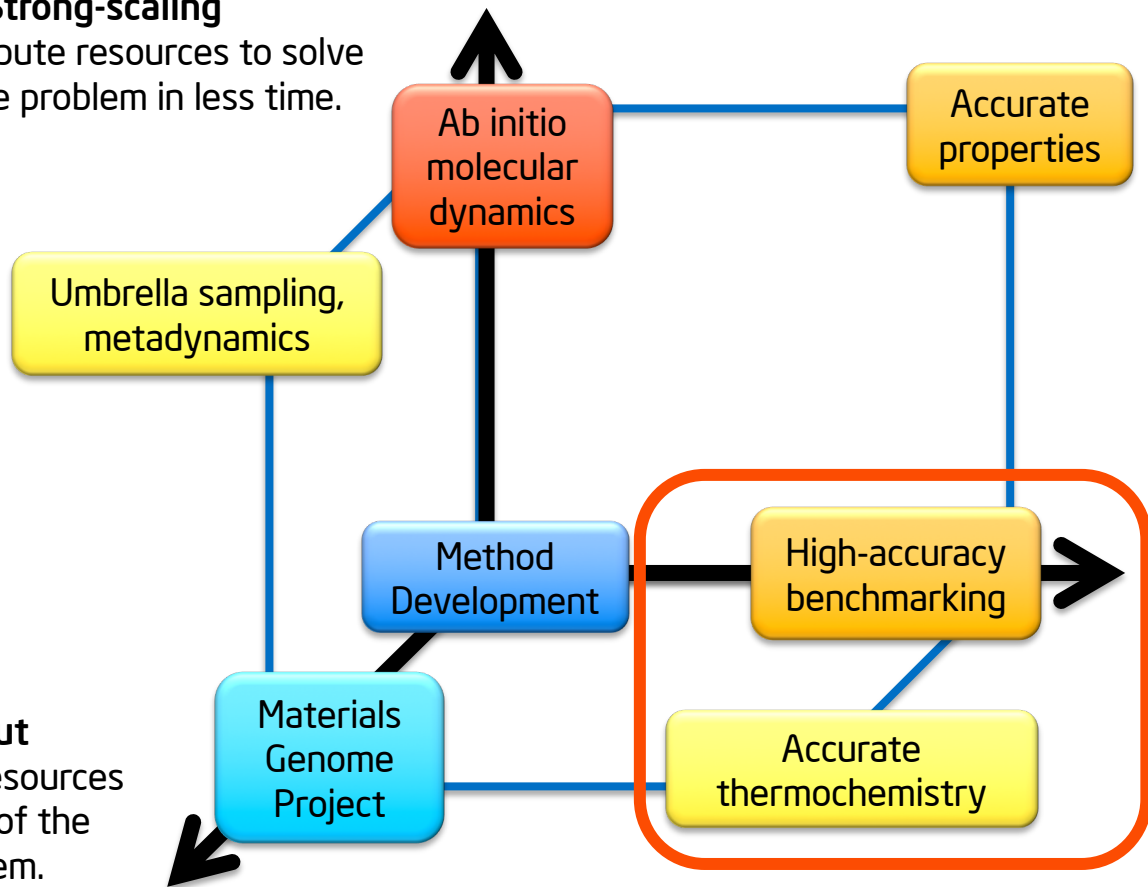


- Suite of computational chemistry functionality:
 - From classical MD to AO DFT ... MP2 to CCSD...
 - Multi-scale: QM/MM, embedding
 - NWPW: AIMD code based on MPI
- Massively parallel design for HPC systems circa ~2000.
 - Process-based parallelism in Global Arrays
 - Modular design to enable reuse of integrals, SCF, etc.
 - Object-oriented design in legacy Fortran
 - Threading from BLAS/LAPACK (until recently)



Strong-scaling

More compute resources to solve the same problem in less time.



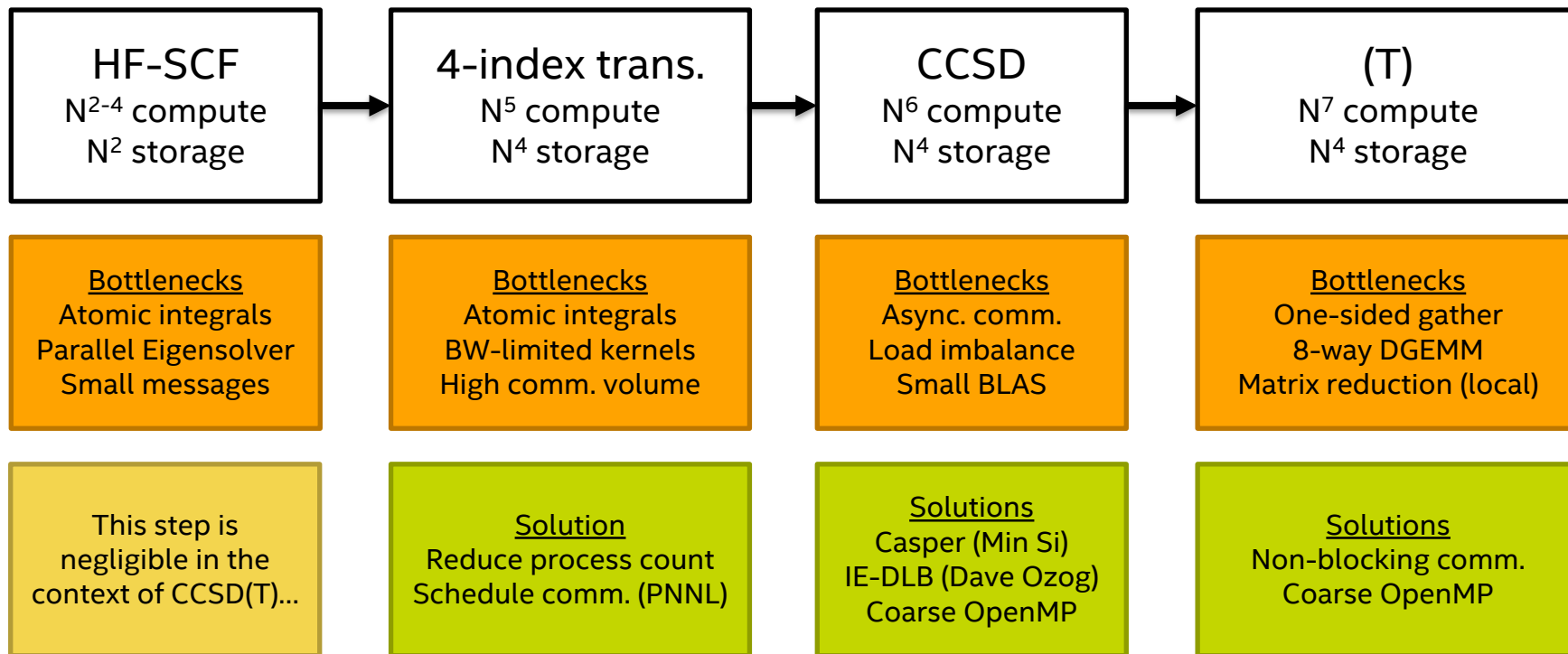
Weak-scaling

More compute resources to solve a larger problem.

Throughput

More compute resources to solve more of the same problem.

Overview of CCSD(T)



OpenMP tradeoffs

- + OpenMP is the **only** (portable) threading model we can use with Fortran.
- + Threads ameliorate memory capacity issues. Replicated data eliminates communication bottlenecks in irregular algorithms (e.g. Fock build).
- + Reducing the process count improves scalability of communication-intensive steps (e.g. global transpose).
- + Increasing compute per process decreases in-cast problem of DLB.
- *Reduces* parallelism because NWChem is fully process-parallel but OpenMP coverage is limited.
- Work/data decomposition not designed with threads in mind.
- Essential components of NWChem are not thread-safe ☹️

Semi-direct CCSD(T) optimizations

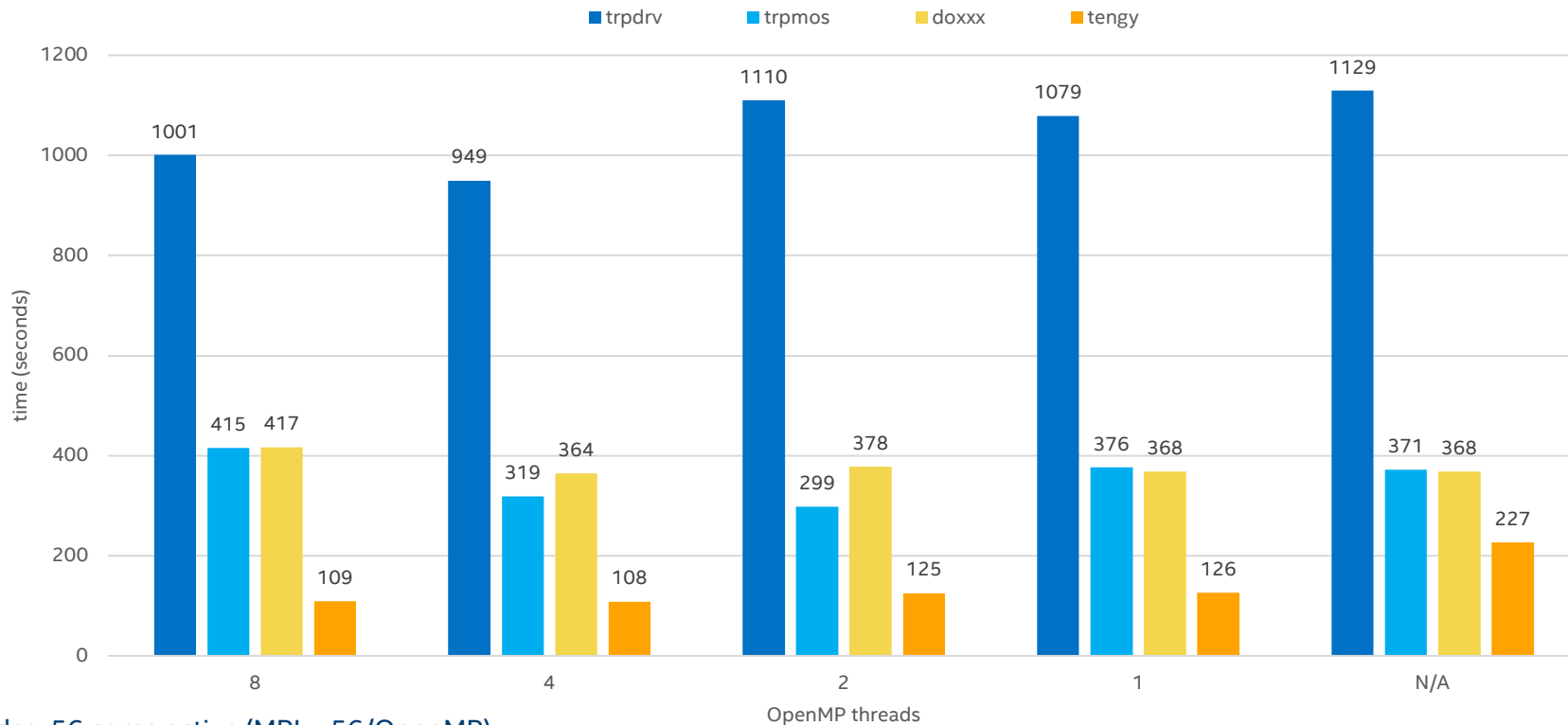
(T) Is ~80% of the total time for large jobs...

1. (T): Make all Get operations nonblocking.
2. (T): Manually inline subroutines and implement one fork-join per iteration.
3. (T): Improve compute intensity in exchange for infrequent extra work (<1%)
4. *CCSD: thread all the important loops.*
5. *CCSD: fuse parallel regions as much as possible.*

OpenMP coverage limited by thread-unsafe atomics integral routines.
We use OpenMP mutual exclusion for all GA calls.

Triples performance

(H₂O)₇ with cc-pVTZ (406 basis functions)
Xeon Platinum 8180 processors (2x28)
Omni Path interconnect, local SSD scratch
Intel Fortran, C/C++, MKL (2018.2.199)



4 nodes, 56 cores active (MPI = 56/OpenMP)

Optimization Notice

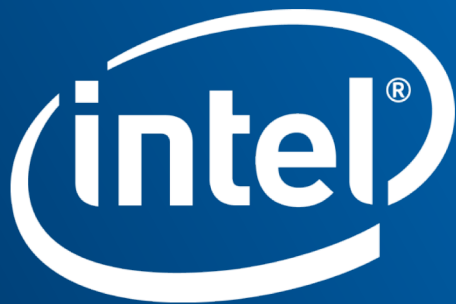
Copyright © 2015, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



Lessons learned

- (T) can run with 8x fewer processes without losing efficiency.
- CCSD does not benefit from OpenMP for problem sizes considered.
- The conversion of (T) from OpenMP host to target was mechanical for KNC.
 - Focused on Xeon Phi coprocessors so performance optimization is more similar to host code than in other cases.
- Semidirect code aligned with traditional OpenMP usage but TCE CCSD is implicitly task-based and will use OpenMP tasks (compiler support limited).
- Thread-safe GA/ARMCI is essential. Localizing the mutual exclusion of GA calls is painful. (ARMCI-MPI can be thread-safe; PNNL GA/ARMCI is WIP)



References

- R. F. Van der Wijngaart, A. Kayi, J. R. Hammond, G. Jost, T. St. John, S. Sridharan, T. G. Mattson, J. Abercrombie, and J. Nelson. ISC 2016. *Comparing runtime systems with exascale ambitions using the Parallel Research Kernels.*
- E. Georganas, R. F. Van der Wijngaart and T. G. Mattson. IPDPS 2016. *Design and Implementation of a Parallel Research Kernel for Assessing Dynamic Load-Balancing Capabilities.*
- R. F. Van der Wijngaart, S. Sridharan, A. Kayi, G. Jost, J. Hammond, T. Mattson, and J. Nelson. PGAS 2015. *Using the Parallel Research Kernels to study PGAS models.*
- R. F. Van der Wijngaart and T. G. Mattson. HPEC 2014. *The Parallel Research Kernels.*